A Hybrid Approach Towards Wrapper Induction

Stefan Raeymaekers and Maurice Bruynooghe

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Leuven, Belgium
{stefanr,maurice}@cs.kuleuven.ac.be

Abstract. The approaches to learn wrappers for extraction from semi-structured documents (like HTML documents) are divided into string based ones, and tree based ones. In previous papers we have shown that tree based approaches perform much better and need less examples than string based approaches, but have the disadvantage that they can only extract complete text nodes, whereas string based approaches can extract within text nodes. In this paper we propose a hybrid approach that combines the advantages of both systems. We compare this approach experimentally with a string based approach on some sub node extraction tasks.

1 Introduction

Wrappers that extract information from web pages are very useful to process data that is only available as a HTML document. The induction of wrappers from examples is an active research field, as manually crafting these wrappers is a tedious job. Moreover, they regularly require maintenance as a change in the templates of a site often invalidates the wrappers.

In a string based approach [1–6, 9, 10, 13], the document is mostly viewed as a sequence of tokens and markup tags from which a subsequence gets extracted. This is done by learning to recognize the *start* and *end* boundaries of the target substring. These boundaries are always between two tokens.[1] The markup tags define an implicit tree structure on the document. In the string representation, the relations in this tree are hidden, as in the flattened tree, parent or sibling nodes of a given node are separated by the tokens and tags that make up the subtrees under its (preceding) siblings. This renders the induction task more difficult.

Tree based approaches [8, 12] view the document as a tree, preserving the tree relations. In [11] we compare some state of the art string based approaches with state of the art tree based approaches and conclude that the latter are much more performant. They need less examples to induce a perfect wrapper, and the induction time is often orders of magnitudes lower. A limitation of these methods though is that they operate on the nodes of the tree and hence can only extract complete nodes. Tasks that require sub node extraction are out of scope. The common reply on this issue is that the tree based approach is a first step in a two level approach in which the sub node extractions are performed in a second step.

In this paper we show that this is indeed a viable approach. We investigate ways to extend a tree based approach with a string based approach as a second step. We

[1] Using characters instead of tokens is an overkill, as the target values do usually not contain half a token, and with tokens, the higher granularity speeds up the learning phase.

implemented one possibility and compared this hybrid approach experimentally with a state of the art string based approach.

As string based approach we have chosen the STALKER system [10]. This system reaches relatively good results by adopting a hierarchical approach. The learning task is split up in simpler tasks that are learned separately. We use the same system, without the hierarchical approach as an extension to the tree based system. There is no need for a hierarchical approach as the sub node learning task in the second step, are most of the times easier then the top level tasks in the hierarchical approach.

As tree based approach we use our system [12] based on (k, l)-contextual tree languages. In [11], an extended version of [12] this system is shown to have superior performance over both state of the art string and tree based systems.

The rest of the paper is organized as follows. In Section 2, we give an overview of the STALKER system, the string based approach we have chosen. Section 3 contains a summary of our wrapper induction with (k, l)-contextual tree languages. In Section 4 we discuss the main contribution of this paper, the extension to enable sub node extraction. Section 5 describes the experimental setup and relates the results. We conclude in Section 7. Below we introduce Example 1. This example will be used as a running example in the coming sections.

Example 1. A web site running a restaurant guide, allows for a search for restaurants based on parts of their name. The resulting list of restaurants is returned as a web page constructed from a fixed template. In Figure 1, a possible outcome is shown for a search on 'china'. From this web page we can extract the following fields: the name of the restaurant(N), its type(T), the city(C) where it is located, and a phone number(P). For each restaurant we could also extract the url(L) from the link (leading to more detailed address information). And from the top sentence, the search term(S) that generated the page can be found. Note that the occurrence of the search term in the name is rendered in italic, while the land code of the phone number is in bold.

2 String Based Approach

In this section we describe the STALKER system [10]. We start with explaining its hierarchical approach. Then we give the semantics of the extraction rules, and we conclude with the induction algorithm.

2.1 Hierarchical Extraction

In contrast with other string based methods, STALKER implements a hierarchical extraction approach. An *Embedded Catalog (EC)* describes the structure of the data. This is a tree structure where the leaves are fields, and the internal nodes either tuples or lists. Figure 2 shows the EC for Example 1. Note that the EC formalism might not be expressive enough to represent some more complex data structures. To extract a specific field, first the parent has to be extracted, and the extraction rules are then applied on the subsequence extracted for the parent. To extract the values of the 'City' field of Example 1, first the subsequence containing the search term and the list of restaurants is extracted.

a)
```
<html><body>
   <b>Restaurant Guide: search results for <i>china</i></b>
   <p><a>New<i>China</i>Town (chinese)</a>
      Brussels<b>Tel: +32</b>(0)2 345 67 89</a></p>
   <p><a>Royal<i>China</i>(chinese)</a>
      Leuven<b>Tel: +32</b>(0)16 61 61 61</a></p>
   <p><a><i>China</i>Garden (chinese)</a>
      Amsterdam<b>Tel: +31</b>(0)20-4321234</a></p>
</body></html>
```

b)

Fig. 1. Restaurant Guide (Example 1): a) HTML code; b) screen shot.

Then the complete list of restaurants is extracted. Then the individual restaurants are extracted. And finally from the subsequences for each restaurant, the 'City' field is extracted. The advantage of this approach is that complex extraction tasks are split into easier problems. Disadvantages are that more examples are needed to learn rules for every level of the hierarchy[2], and that errors in the different levels will accumulate.

2.2 Rules

To extract a subsequence from a sequence of tokens, the STALKER system uses a start and an end rule, to find the boundaries of that subsequence. The start rules are executed in forward direction from the beginning of the sequence, the end rules are executed in backward direction. A STALKER rule is either a simple rule or a disjunction of simple rules. In the latter case the boundary is given by the first simple rule that does not fail. A simple rule is a list of so-called landmarks. A landmark is a sequence pattern consisting of tokens and/or wildcards. On execution, the rule searches for a part of the sequence that matches the first landmark. From the end of this part the search for the second landmark is started, and so on. The boundary that is finally returned is either the end or the beginning of the part that matched the last landmark. Which one is indicated by a modifier; SkipTo or SkipUntil for respectively the end or the beginning (or BackTo and BackUntil for rules in the other direction). When the search for a landmark reaches the

[2] To learn list extraction, each example should consist of two consecutive elements of the list.

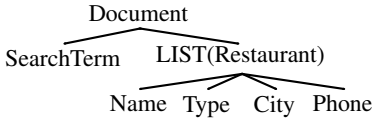


Fig. 2. Embedded Catalog for the restaurant guide example (Example 1).

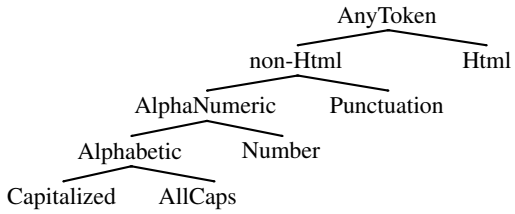


Fig. 3. Wildcard hierarchy. A token that matches a wildcard of a given type, will also match the wildcards of the ancestors of that type.

end/beginning of the sequence, the rule is said to fail. STALKER uses multiple types of wildcards that form a type hierarchy. This hierarchy is shown in Figure 3.

Example 2. Consider the first subsequence extracted for the tuple 'Restaurant':

```
New <i> China </i> Town ( chinese ) </a> Brussels <b>
Tel : + 32 </b> ( 0 ) 2 345 67 89
```

The rule *SkipTo(*`</a>`*)* applied on this sequence returns the position at the end of the first occurrence (and single occurrence in this example) of the tag '`</a>`', hence at the beginning of 'Brussels'. The rule *BackTo(*`<b>`*)* goes backward and returns the position at the end of 'Brussels'. Hence these rules are a start and end rule for the 'City' field.

For the rule *SkipUntil(*AnyToken*)* we see that the first token of the restaurant sequence is matched by the wildcard 'AnyToken'. As the modifier is 'until', the beginning of that token is returned. This is the beginning of 'New' for the above sequence. The rule *BackTo(*`</a>`*) BackTo(*`'('`*)* goes backward to the position before the first matching '`</a>`' token, and then continues going backward from there on until the first `'('` encountered. The position between 'Town' and '(' will be returned. Therefore these rules can be used to extract the 'Name' field.

To extract the sub-sequences for the tuple 'Restaurant' from the list of restaurants (or the sequence extracted for that list), we use the startrule and endrule repeatedly. The first start boundary though coincides with the start boundary of the list (and the last end boundary coincides with the end boundary of the list). The startrule *SkipTo(*`<p><a>`*)* returns the position at the end of the first occurrence of these two consecutive tags. The endrule for this extraction task is: *BackTo(*`</p>`*)*.

2.3 Induction Algorithm

The STALKER induction algorithm starts from a set of positive examples (each consisting of a sequence wherein boundaries of a subsequence are given). As long as this set is not empty, a new simple rule is learned, those examples covered by this rule are removed from the set, and that rule is added to the disjunction of rules that will be the final result. The algorithm to learn a simple rule chooses one *seed* example (the shortest example in the set) to guide the induction, the other examples are used to test the quality of candidate rules. The algorithm does not search the entire rule space for the best rule.

In each loop it takes two rules from a given set of rules, one is the best solution in that set, the other is the best refiner. Some heuristic rules are designed to define a ranking (best solution and best refiner) over a set of rules. This ranking is based on properties of the rules, and on the number and quality of the extractions of each rule on the other examples. The refinements of the best refiner, together with the best solution gives the new rule set for the next iteration. This loop continues until a perfect solution is found (one that either extracts correctly from an example or fails on that example) or until all refinements fail. The initial set of candidate rules are single landmark rules, with each landmark a single token or wildcard (occurring in the seed). The refinement step will either extend one of the landmarks of a rule with an extra token or wildcard (the extended landmark has to match within the seed), or add a new single token/wildcard landmark somewhere in the rule (the token or wildcard has to occur in the seed).

3 Tree Based Approach

In this section we define the notion of (k,l)-Contextual Tree Languages, as a subclass of the regular tree languages. In contrast with the whole class of regular languages, this subclass can be learned from positive examples only. The intuition behind (k,l)-contextual tree languages is fairly straightforward. At the base is a parameterized deconstruction of a tree into its building blocks called (k,l)-forks. These are subparts of the tree with maximally k consecutive children and a maximal depth of l. A tree belongs to a given language iff its (k,l)-forks all belong to the representative set of building blocks for that language. To learn a (k,l)-contextual tree language from examples, the (k,l)-forks of these examples are collected into a representative set for the learned language.

We start with formal definitions of (k,l)-forks and (k,l)-contextual tree languages. We then show how tree languages can be used to represent wrappers, and how to learn the parameters.

3.1 Preliminary Definitions

An alphabet Σ is a finite set of symbols. The set $T(\Sigma)$ of all finite, unranked trees with nodes labelled by elements of Σ can be recursively defined as $T(\Sigma) = \{f(w) \mid f \in \Sigma, w \in T(\Sigma)^*\}$. We denote $f(\epsilon)$, with ϵ the empty sequence, by f. The subtrees of a tree are inductively defined as $sub(f(t_1,\ldots,t_n)) = \{f(t_1,\ldots,t_n)\} \cup \bigcup_i sub(t_i)$. A *tree language* is any subset of $T(\Sigma)$. The set of *(k,l)-roots* of a tree $f(t_1,\ldots,t_n)$ is the singleton $\{f\}$ if l=1; otherwise, it is the set of trees obtained by extending the root f with $(k,l-1)$-roots of k successive children of t (all children if $k > n$). Formally:

$$R_{(k,l)}(f(t_1,\ldots,t_n)) = \begin{cases} \text{if } l=1 \text{ then } \{f\} \\ \text{if } l>1 \text{ and } k>n \text{ then } f(R_{(k,l-1)}(t_1),\ldots,R_{(k,l-1)}(t_n)) \\ \text{else } \bigcup_{p=1}^{n-k+1} f(R_{(k,l-1)}(t_p),\ldots,R_{(k,l-1)}(t_{p+k-1})) \end{cases}.$$

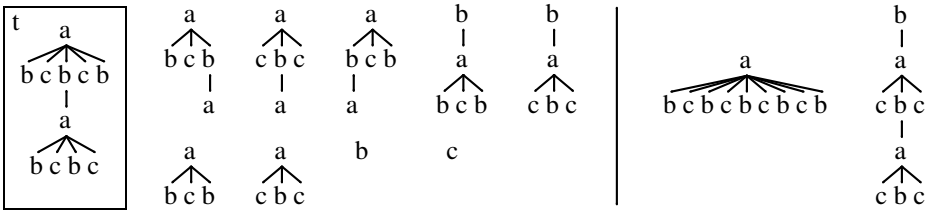
In this formula, $f(S_1,\ldots,S_n)$, denotes the set $\{f(s_1,\ldots,s_n) \mid s_i \in S_i\}$. In a similar notational extension, $R_{(k,l)}(T)$ denotes $\bigcup_{t\in T} R_{(k,l)}(t)$, the (k,l)-roots of a set T of trees. Finally, a (k,l)-*fork* of a tree t is a (k,l)-root of any subtree of t. Thus, the set of (k,l)-forks of t can be written as $R_{(k,l)}(sub(t))$ and we denote it by $F_{(k,l)}(t)$. Then the (k,l)-forks of a set of trees T are defined as $F_{(k,l)}(T) = \bigcup_{t\in T} F_{(k,l)}(t)$.

3.2 (k,l)-Contextual Tree Languages

Definition 1. *The (k,l)-contextual tree language based on the set G of trees is defined as* $L_{(k,l)}(G) = \{t \in T(\Sigma) \mid F_{(k,l)}(t) \subseteq G\}$.

As shown in [12], the language $L_{(k,l)}(F_{(k,l)}(E))$ is the most appropriate (k,l)-contextual tree language that can be learned from a set of positive examples E as it is the most specific (k,l)-contextual language that accepts all the examples. Generalization is controlled by the choice of the parameters; they determine the minimal granularity of the building blocks (the forks from the examples) that can be used in defining the language. Negative examples can be used to adjust the parameter values [12].

Example 3. Below we show graphically the $(3,3)$-forks of a tree t. The first three of these forks are the $(3,3)$-roots of t. Two trees from the language $L_{(3,3)}(F_{(3,3)}(\{t\}))$ are shown on the right.



3.3 Wrapper Induction

A marking of a tree $t \in T(\Sigma)$ is a function that maps a tree on a marked version of that tree $t' \in T(\Sigma_X)$, by replacing some of its nodes s with a marked equivalent s_X. The marked alphabet Σ_X is defined as $\Sigma_X = \Sigma \cup \{s_X \mid s \in \Sigma\}$. A correctly marked tree (with regard to the extraction task) is defined as the single marked version of a tree in which all target nodes, and no others, are marked, while a partially correct marked tree requires that only some of the target nodes, and no others, are marked. We represent our wrapper as a language that accepts only partially correct marked trees. During extraction, a node is extracted if after marking that single node the resulting tree is accepted by the wrapper language. The wrapper is learned from examples that consist of the document tree with exactly one of the target nodes marked.

In [11], (k,l)-contextual tree languages are used for the correct marking acceptor. To enhance the generalization power of the algorithm, a preprocessing of the text nodes and a filtering of the forks is added. During the preprocessing step, the text nodes (except elements of the distinguishing context) are replaced by wildcards. The use of distinguishing contexts is optional, and the set of contexts is learned from the given examples. Only the marked forks are used; they provide the local context needed to decide whether a node should be extracted or not, while the other forks describe the general structure of the document. The latter is not needed as we assume that all documents for a given task are generated from the same template.

Example 4. In Figure 4.a we show the tree (only a subtree due to space restrictions) of the document from Figure 1, with the target fields indicated beneath. Only the 'City'

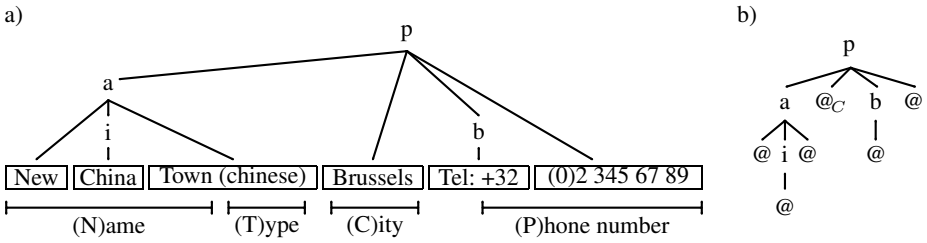


Fig. 4. a) A subtree of the document from Figure 1, containing the first restaurant. The different fields are indicated below the text leaves. b) The same subtree preprocessed, with the target node of the (C)ity field marked.

field can be extracted with the regular tree based approach, as it is the only one that occupies a single text node. Given the value 'Brussels' as example element of that field, we will mark the node containing 'Brussels', and perform the preprocessing step. The result (for the subtree of Figure 4.a) is shown in Figure 4.b. The learning algorithm collects now the (k, l)-forks that contain the marker from this tree. For $k = 2$ and $l = 1$, the resulting set is $\left\{ @_C\,,\ \begin{array}{c} \text{p} \\ | \\ @_C \end{array} \right\}$, for $k = 2$ and $l = 2$, it is $\left\{ @_C\,,\ \begin{array}{c} \text{p} \\ \text{a}\ \ @_C \end{array},\ \begin{array}{c} \text{p} \\ @_C\ \ \text{b} \end{array} \right\}$. The first wrapper will extract 'Brussels' and '(0)2 345 67 89' from the subtree in Figure 4.a, hence it is to general. The second wrapper extracts only 'Brussels' and is therefore a correct marking acceptor.

Another 'single node' field is the '(S)earch term'. A correct marking acceptor is obtained with the parameters $k = 1$ and $l = 3$: $\left\{ \begin{array}{ccc} & & \text{b} \\ & \text{i} & | \\ @_S\ , & | & \text{i} \\ & @_S & | \\ & & @_S \end{array} \right\}$.

The induction algorithm is able to learn from positive examples only. But suitable values for the parameters k, l, and a boolean to turn the distinguishing contexts on or off have to be specified. Smaller values lead to more general acceptors, while larger values result in more specific ones. To obtain correct marking acceptors, while avoiding overfitting, one can search for the most general wrapper that rejects a set of given negative examples. In [12] an efficient algorithm is given to search through the parameter space.

All the above is integrated in an interactive system that starts induction from a single positive example. During the interaction, a user can apply the current wrapper on a document and can provide a negative example by selecting a false positive and a positive example by selecting a false negative. The wrapper is updated after each new example.

4 Hybrid Approach

This section describes how our tree based approach can be combined with a string based approach for extraction of fields that do not coincide with a single node.

4.1 Sub Node Fields

The term sub node field means that the text value from the field does not necessarily begin or end at a node boundary. The value can be a substring of a single text node, or could start and end in different nodes (the boundary nodes) in which case the value is the accumulation of the strings in the text nodes between the boundary nodes.

We define a spanning node for a given occurrence of a field as the first common ancestor of the two boundary nodes. In the case that the start node and end node are the same node, the spanning node is defined to be this node itself. These two cases are represented schematically respectively in Figure 5.a and b. We will refer to them as cases a and b. As can be seen in Figure 4, the 'Name' field is an example of case a, with as spanning node the 'a' node that spans over the start and end node. The 'Type' field in the same figure is an example of case b. Note that the boundary nodes are not necessarily at the same depth in the tree, as illustrated by the 'Phone' field. For this occurrence the spanning node is the 'p' node.

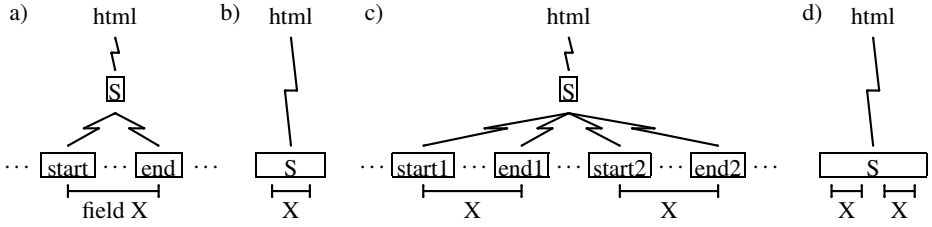


Fig. 5. Schematic representation of the different possible configurations in which an occurrence of a field can be found in a tree. The broken lines indicate an ancestor relation of one or more levels deep (The intermediate (irrelevant) nodes are left out).

In Example 1, all occurrences of a same field have different spanning nodes. It is possible though that different occurrences share the same spanning node. This case (case c) is represented in Figure 5.c. This case can also degenerate such that the boundary nodes and spanning node coincide. Hence multiple field values can be extracted from a single text node. This is illustrated in Figure 5.d, and we refer to it as case d.

4.2 Possible Approaches

We take two approaches to combine the tree based node extraction with a token sequence based subsequence extraction. A first approach is to extract (or learn to extract) the spanning node, and then extract (or learn to extract) the correct subsequence from the sequence obtained by flattening the subtree that starts at the spanning node. From this sequence we remove the initial (before the first text node) and trailing (after the last text node) mark up tags.

Example 5. In Example 2, the Name, Type, City, and Phone fields are all extracted from the sequence that is extracted for the Restaurant tuple (the previous level in the

hierarchy). In the first approach the spanning node 'a' is extracted, and sequence based extraction is then performed on the sequence defined by this spanning node:

```
New <i> China </i> Town ( chinese )
```

This sequence is smaller than in the hierarchical STALKER approach. The end rule *BackTo(*`'('`*)* suffices, as opposed to the *BackTo(*`</a>`*) BackTo(*`'('`*)* rule given in Example 2. For the Type field, the sequence is even smaller. The spanning node is a text node (case b). The City field simplifies to node extraction, no sequence extraction is needed. Only the Phone field, with spanning node 'p' will need sequence extraction from the same sequence as in the hierarchical STALKER approach. For the other fields, extraction and rule induction are performed on smaller sequences, leading to smaller and more correct rules, and a faster induction.

The second approach is to perform two node extraction tasks, one for the start node, and one for the end node. In a second step, the start boundary is retrieved from the start node, and the end boundary from the end node. Although two different sequences are used in this approach, these sequences are in general smaller than in the first approach.

On a case by case basis, we see that for case a, the second approach is better, because the sequence will perform better on smaller sequences. For case b, the second approach is overkill as there is no need to extract the same node twice. In case c and d, the first approach will not suffice with a single level sequence extraction. We need to use a limited hierarchical extraction that will do a list extraction of the multiple field values under the single spanning node. For case c, the second approach will be able to extract the different start and end nodes separately, and will not need a hierarchical sequence extraction. However, for extracting all targets in case d, the second approach also requires the use of hierarchical sequence extraction.

Overall, the second approach seems to be the preferable one. But having a look at real world extraction tasks, it turns out that Example 1, having two fields in case a, is a bit contrived. Indeed, the overwhelming majority of extraction tasks we looked at is either case b, or single node extraction without the need for sequence extraction. Extraction tasks situated in case c and d occur rarely. Hence the first approach is not too bad after all.

As the goal of the paper is to explore the viability of a hybrid scheme, it is sufficient to implement one approach and to compare it with the hierarchical STALKER system. We have chosen for the first approach, as it is a more straightforward extension to our existing system that is already able to extract a single node, so we only had to add a postprocessing step.

4.3 Interactive System

We have extended the system (that has a GUI), described in [12]. Instead of initially clicking on a single text node, the user selects a subsequence as the initial positive example. The system enters a loop in which it interacts with the user to improve the wrapper, until the user is satisfied. In each iteration, it induces a hypothesis based on the given positive and negative examples. The extraction results for the current document are visualized, and the user is invited to give counterexamples when the hypothesis is

not perfect. For false negatives, the user simply selects a *new positive example*. For false positives we distinguish two cases. Either an extraction is shown at a correct position, but the extraction itself is too big or too small. The user can then select the correct extraction, providing a *correction* to the system. Or the extraction is at a position without a target value in the neighborhood. The user can indicate this, providing a *new negative example*.

Internally, the spanning nodes of the example selections (both new positive examples and correction) are retrieved to find the set of positive node examples. The spanning nodes of the rejected extractions are collected to form the set of negative nodes. Based on these two sets, the induction algorithm for (k, l)-contextual languages, learns a set of parameters and the associated marked tree language for the node extraction.

Next, for a (new) positive example, the sequence under the spanning tree together with the selected field provides a (new) example for the STALKER induction algorithm.

Note that a new negative example requires only to learn again the extraction of the spanning node as the set of examples used by STALKER is preserved, given that STALKER only uses positive examples. A correction, on the other hand, will often not affect the position of the spanning node, in which case it only provides a new example for STALKER.

5 Experiments

In our experimental setup we want to compare the number of interactions by the user needed to learn a correct wrapper. For hierarchical STALKER this means that for every level the correct rules have to be learned. For every level the user has to give two initial examples, and extra corrections until no more mistakes can be found. For the hybrid approach, the user has to give a single initial example, and as many false positives, false negatives, and corrections as needed to learn a perfect wrapper. To simulate the user, we choose the annotated training set to find all mistakes, and take a random one to pass to the learning algorithm.

We use the WIEN data sets[3] for our comparison. We only used the sets that have a set of annotations included in the repository, and we left out some that were hard to represent in the STALKER embedded catalog formalism. Every data set has multiple fields. As we compare a single field extraction task. We split the tuple extraction task for every data set into several single field extraction tasks. Each task is referred to with the name of the original data set combined with the index of the field in the tuple. Some fields are contained in the 'href' attribute of an 'a' tag, or the 'src' attribute of an 'img' tag. In the tree based approach, the HTML-parser associates the attributes to the corresponding node. A trivial step can be added to retrieve these values. We decided to leave these tasks out, as they are skewed in favour of the tree based approach.

In Table 1 we show the averaged results of 30 runs on each data set. We give the induction time for the two approaches in column *ms*. For the hybrid approach we give the final k and l values, and the number of **P**ositive examples, **N**egative examples, and **C**orrections. For the hierarchical STALKER approach, we show the number of **P**ositive

[3] These are available at the RISE repository: http://www.isi.edu/info-agents/RISE/index.html.

examples, split over the different levels (starting on left with the top level). When we compare the total number of interactions (P+N+C and P summed over all levels), it is clear that the hybrid approach requires substantially less user interactions. The sequence extraction step in the hybrid approach, and the extraction in the final level of the STALKER approach extract the same text value. When we compare the number of positive examples needed to learn this last extraction (P+C compared with the last number in P), we see that this number is again smaller for the hybrid approach. This is because the tree based approach returns a much smaller sequence to extract from, as illustrated in Example 5.

Table 1. Comparison of the interactions needed to learn a perfect wrapper, between our hybrid approach, and the a sequence based approach (STALKER).

Data	Hybrid				STALKER	
set	P/N/C	k	l	ms	P	ms
s1-1	1/1/0	1	3	18	3/72.1/2.9	4442
s1-3	4/1.8/0	3	3	612	3/60.6/6.9	3651
s3-2	1/1/0	1	3	10	2.9/2.1/2.2	460
s3-3	1/0/0	1	2	3	2.5/2.1/2.3	316
s3-4	1/0/1.2	1	2	6	2.8/2.1/3	394
s3-5	1/0/4.9	1	2	48	2.8/2.1/5.4	554
s3-6	1/0/3.4	1	2	7	2.9/2/6.1	27520
s4-1	1/0/0	1	2	2	4.5/2.2/2.3	1136896
s4-2	1/1/0	2	3	33	4.7/3/2.1	1240828
s4-3	1/1/1	2	3	23	4.7/2.2/2	1420509
s4-4	1/1/1	2	3	22	4.8/2.7/2	1333724
s5-2	1/1/0	1	4	18	2.8/3.7/2.9	1136
s8-2	1/1/0	1	3	12	2.4/2/2.6	785
s8-3	1/1.2/0	2	3	39	2.3/2.1/2.9	675
s12-2	2/1.4/0	1	4	36	2.7/80.6/2.3	1394
s14-1	1.1/1/0.9	2	2	21	2/2.3/2.3	21
s14-3	1/0/0	1	2	3	2/2.2/2.4	21
s15-2	1/0/0	1	2	2	2.9/2.1/2.1	5
s19-2	1/1/0	2	2	13	2/2.1/2.1	85
s19-4	1/1/0	1	3	7	2/2/2	85

Data	Hybrid				STALKER	
set	P/N/C	k	l	ms	P	ms
s20-3	1/0/0	1	2	3	2/2/2.1	155
s20-4	1/1.4/0	2	3	192	2/2/3	165
s20-5	1/1.8/0	2	3	1067	2/2/2	158
s20-6	1/1.4/0	2	3	41	2/2/3.1	159
s23-1	1/1/0	2	3	35	2.6/3.1/4.1	606
s23-3	1/1/0	1	3	11	2.6/2.6/3.4	602
s25-2	1/1/0	1	3	5	2.6/5.1/3.0	82
s27-1	1/1.6/1	2	6	195	2.7/2.4	44
s27-2	1/1.3/1	2	6	190	2.7/2.8	46
s27-3	1/1.4/1	2	6	235	2.7/2.8	52
s27-4	1/1.2/1	2	6	348	2.6/6.7	3430
s27-5	1/1/1	2	5	125	2.7/2.5	33
s27-6	1/1/0	2	5	101	2.9/2.3	44
s30-2	2/1/0.7	1	3	14	2/2.8/2.6	8
s30-3	2/1/0	1	3	14	2/3/2	8
s30-4	2/1/0	2	2	50	2/3.3/2.5	12
s30-5	2/1/0.2	2	2	26	2/2.7/2.1	7
s30-6	2/1/0	2	2	49	2/3.1/2.8	9
s30-7	2/1/0	2	2	28	2/2.5/2.1	6
s30-8	2/1/0	2	2	25	2/2.6/2.6	6

6 Related Work

Another approach that allows to combine sequence based and tree based methods is described in [7]. A (set covering) meta learning algorithm runs the learning algorithms of different wrapper modules, evaluates their results and chooses the best resulting rules to add to the final solution. Some of these modules are defined to combine other modules to allow conjunctions or a multi level approach like ours. In contrast to our approach, the algorithm requires completely annotated documents (or at least a completely annotated part of the document).

7 Conclusion

Tree based methods have been shown to have a favorable performance over string based ones, on complete node extraction tasks, but have as limitation that they cannot extract values that have boundaries within text nodes [11]. In this paper we show that these methods are viable, when used as a first step, in a hybrid two step approach. We have shown that the tree based approach does a good job of narrowing down the sub node extraction task presented to the string based second step. This results in substantially faster learning while requiring substantially less user interactions.

The hybrid approach as presented in this paper will extract only a single field, instead of n-tuples (in contrast, STALKER can extract n-tuples as long as they can be represented in the embedded catalog formalism). Also in [11] we argue in a section about further work that it is more flexible to add a tuple aggregation procedure on top of a single field extraction approach. A practical approach and an empirical proof of the viability of this last approach still remains for further work.

References

1. M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI/IAAI*, pages 328–334, 1999.
2. B. Chidlovskii, J. Ragetli, and M. de Rijke. Wrapper generation via grammar induction. In *Proc. 11th European Conference on Machine Learning (ECML)*, volume 1810, pages 96–108. Springer, Berlin, 2000.
3. D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *AAAI/IAAI*, pages 517–523, 1998.
4. D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Innovative Applications of AI Conference*, pages 577–583. AAAI Press, 2000.
5. D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
6. C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.
7. L. S. Jensen and W. W. Cohen. A structured wrapper induction system for extracting information from semi-structured documents. In *Proc. of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.
8. R. Kosala, M. Bruynooghe, H. Blockeel, and J. V. den Bussche. Information extraction from web documents based on local unranked tree automaton inference. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 403–408, 2003.
9. N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
10. I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.
11. S. Raeymaekers, M. Bruynooghe, and J. V. den Bussche. Learning (k,l)-contextual tree languages for information extraction. Submitted to Machine Learning Journal.
12. S. Raeymaekers, M. Bruynooghe, and J. V. den Bussche. Learning (k, l)-contextual tree languages for information extraction. In *ECML*, pages 305–316, 2005.
13. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.